Computer Physics Communications 261 (2021) 107810

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

PyXtal: A Python library for crystal structure generation and symmetry analysis☆,☆☆

Scott Fredericks, Kevin Parrish, Dean Sayre, Qiang Zhu *

Department of Physics and Astronomy, University of Nevada Las Vegas, Las Vegas, NV 89154, USA

ARTICLE INFO

Article history:
Received 26 November 2019
Received in revised form 11 December 2020
Accepted 17 December 2020
Available online 31 December 2020

Keywords:
Symmetry
Crystallography
Structure prediction
Wyckoff sites
Global optimization
Phase transition

ABSTRACT

We present PyXtal, a new package based on the Python programming language, used to generate structures with specific symmetry and chemical compositions for both atomic and molecular systems. This software provides support for various systems described by point, rod, layer, and space group symmetries. With only the inputs of chemical composition and symmetry group information, PyXtal can automatically find a suitable combination of Wyckoff positions with a step-wise merging scheme. Further, when the molecular geometry is given, PyXtal can generate different dimensional organic crystals with molecules occupying both general and special Wyckoff positions. Optionally, PyXtal also accepts user-defined parameters (e.g., cell parameters, minimum distances and Wyckoff positions). In general, PyXtal serves three purposes: (1) to generate custom structures, (2) to modulate the structure by symmetry relations, (3) to interface the existing structure prediction codes that require the generation of random symmetric structures. In addition, we provide several utilities that facilitate the analysis of structures, including symmetry analysis, geometry optimization, and simulations of powder X-ray diffraction (XRD). Full documentation of PyXtal is available at https://pyxtal.readthedocs.io.

Program summary

Program Title: PyXtal
CPC Library link to program files: https://doi.org/10.17632/wfyxyhjzwx.1
Licensing provisions: MIT [1]
Programming language: Python 3
Nature of problem: Knowledge of structure at the atomic level is the key to understanding materials' properties. Typically, the structure of a material can be determined either from experiment (such as X-ray diffraction, spectroscopy, microscopy) or from theory (e.g., enhanced sampling, structure prediction). In many cases, the structure needs to be solved iteratively by generating a number of trial structure models satisfying some constraints (e.g., chemical composition, symmetry, and unit cell parameters). Therefore, it is desirable to have a computational code that is able to generate such trial structures in an automated manner.
Solution method: The PyXtal package is able to generate many possible random structures for both atomic and molecular systems with all possible symmetries. To generate the trial structure, the algorithm can either start with picking the symmetry sites randomly from high to low multiplicities, or use sites that are predefined by the user. For molecules, the algorithm can automatically detect the molecules' symmetry and place them into special Wyckoff positions while satisfying their compatible site symmetry. With the support of symmetry operations for point, rod, layer and space groups, PyXtal is suitable for the computational modeling of systems from zero, one, two, and three dimensional bulk crystals.
References:
[1] https://opensource.org/licenses/MIT

1. Introduction

Knowing the atomic structure is the key to understanding the properties of materials. Ideally, the full atomic structure can be experimentally determined through single crystal X-ray diffraction. If a single crystal sample is not available, only partial

☆ The review of this paper was arranged by Prof. D.P. Landau.

☆☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: qiang.zhu@unlv.edu (Q. Zhu).

https://doi.org/10.1016/j.cpc.2020.107810
0010-4655/

structural information can be extracted from various characterizations, such as powder X-ray diffraction/absorption, Raman spectroscopy, nuclear magnetic resonance, and electron microscopy. Based on this partial structural information (e.g., symmetry, unit cell), a number of trial structures are constructed and optimized at their corresponding thermodynamic conditions. The simulated pattern for each relaxed structure is then compared with the observed one. By doing this iteratively, the structure can be finally resolved. It has been previously demonstrated that structures can be predicted computationally using first-principles [1,2]. The basic idea of computational structure prediction is to guess the correct crystal structure under specific conditions by computationally sampling a wide range of possible structures via different global optimization techniques (e.g., random search [3], metadynamics [4], basin hopping [5], evolutionary algorithms [6,7], particle swarm optimization [8]). After many attempts, the most energetically stable structure found is the one most likely to exist.

For structure determination from either partial experimental information or pure computation, a number of trial structures are needed. It is generally believed that by beginning with already-symmetric structures, fewer attempts are needed to find the global energy minimum [3,9]. For inorganic crystals, symmetry constraints have been encoded in many computational structure prediction codes such as AIRSS [3], USPEX [9], CALYPSO [10] and XtalOpt [7]. For a given crystal with symmetry, the atomic positions are classified by Wyckoff positions (WP) [11]. Two approaches are used to place the atoms into the Wyckoff sites so that the structure satisfies the desired symmetry. One is to pre-generate a set of WPs and then add atoms to these sites [10, 12,13]. The other is to place atoms to the most general WPs and then merge them to the special sites if there exist close atomic pairs [9,14]. This will be repeated until the desired stoichiometry is achieved. The development of new computational tools has allowed the structures of many new and increasingly complex materials to be anticipated [2].

For the prediction of organic crystals, the role of symmetry is even more pronounced. In the periodically conducted Blind Tests of organic crystal structure prediction organized by the Cambridge Crystallographic Data Centre [15], most research groups attempted to reduce the structure generation to a limited range of space group choices with one molecule in the asymmetric unit (Z'). This is based on a statistical analysis that most organic crystals tend to crystallize in only a few space groups with $Z' = 1$ [16]. Currently, there exist a few free packages [17, 18] which allow the generation of random molecular crystals with $Z' = 1$. Combined with modern structure search algorithms (including quasi-random [19], parallel tempering [20], genetic algorithms [21], and evolutionary methods [14]), one can perform an extensive search for the plausible structures. Their energies can then be evaluated with different energy models from the empirical to ab-initio level. A recent blind test [15] has shown that the combination of effective structure generation and an energy ranking scheme can predict not only the structure of simple rigid molecules, but also the molecules representing real-life challenges.

Despite the fact that many programs have their own built-in functions to generate crystals with specific space groups or clusters with specific point groups, most of these functions are implemented in the main packages and cannot work in a standalone manner. To our knowledge, there is only one open source code Randspg [12] that provides the interface to generate 3D atomic crystal structures. Similarly, most molecular crystal generators only support molecules occupying the general WPs with $Z' = 1$, except for the recent development of Genarris 2.0 [22] which is able to deal with structures having a non-integer value of Z' (meaning molecules can occupy special WPs). So far, there is no single code which enables the generation of molecular crystals with arbitrary Z', varying from fractional to multiple integers. While 90% of organic crystals in the Cambridge Structure Database (CSD) have Z'=1, recent advances in experimental polymorph searching and crystal engineering highlight the rich variety of multi-component crystals (co-crystals, salts, solvates, etc.) as well as crystal structures with multiple molecules in the asymmetric unit. For instance, many well-studied molecules, including aspirin [23], resorcinol [24], coumarin [25], glycine [26], DDT [27], and ROY [28], were found to adopt crystal structures with $Z' > 1$. Lastly, neither Randspg nor Genarris supports the generation of low dimensional crystals, which require explicit consideration of layer/rod/point-group (instead of space-group) symmetry operations. Collectively, these cases motivated us to develop a standalone Python program called PyXtal which can be used for customized structure generation for different-dimensional systems, including atomic clusters and 1D/2D/3D atomic/molecular crystals. In Sections 2 and 3 , we will detail the algorithms and the software dependencies. The basic usages of PyXtal will be introduced in Section 4, followed by two example studies using PyXtal in the context of structure prediction in Section 5. Finally, we summarize the features of PyXtal and conclude the manuscript in Section 6.

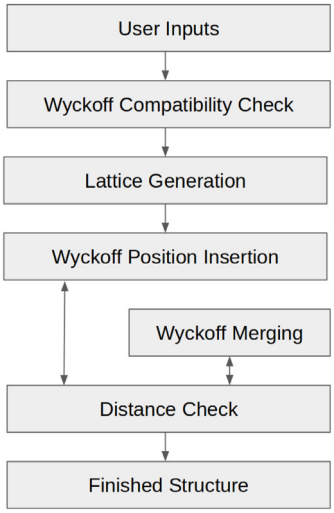

Fig. 1. PyXtal structure generation flowchart. Generation is based on inputs from the user.

2. Algorithms

The core algorithms in PyXtal involve (1) generation of random symmetric crystals and (2) modulation of structures according to the symmetry relation.

2.1. Structure generation

PyXtal adopts the following algorithm to generate a trial structure. First, the user inputs their choice of dimension (0, 1, 2, or 3), symmetry group, stoichiometry, and relative volume of the unit cell. Optionally, additional parameters may be chosen that constrain the unit cell and maximum inter-atomic distance tolerances. Next, PyXtal checks if the stoichiometry is compatible with the choice of symmetry group. If the check passes, trial structure generation begins. Fig. 1 shows a flowchart of the algorithm.

Each step has a maximum number of attempts. If the generation attempt fails at any point, the algorithm will revert progress

for the current step and try again until the maximum limit of attempts is encountered. This ensures that the algorithm stops in a reasonable amount of time while still giving each generated parameter a chance for success. For certain inputs, structure generation may take many attempts or fail after the maximum number of attempts. Typically, these failures indicate that the input parameters are not likely to produce a realistic structure without fine-tuning the atomic positions. In such cases, a larger unit cell volume or a smaller distance tolerance may prevent failure. Below we discuss the technical details implemented during structure generation.

2.1.1. Wyckoff compatibility checking

Before generating a trial structure, PyXtal performs a WP compatibility check. Since WPs in different space groups have different multiplicities, this is a required step that ensures compatibility between a stoichiometry and its assigned space group. For example, consider the space group *Pn-3n* (#222), which has a minimum WP of 2a, followed by 6b. To create a crystal structure with 4 atoms in the unit cell for this symmetry group, the combination of Wyckoff positions must add up to 4. Here, this is not possible. The position 2a cannot be repeated, because it falls on the exact coordinates (1/4, 1/4, 1/4) and (3/4, 3/4, 3/4). A second set of atoms in the 2a position would overlap the atoms in the first position, which is physically impossible.

Thus, from our previous discussion, it is necessary to check the input stoichiometry against the WPs of the desired space group. PyXtal implements this by iterating through all possible combinations of WPs within the confinements of the given stoichiometry. As soon as a valid combination is found, the check returns True. Otherwise, if no valid combination is found, the check returns False and the generation attempt raises a warning.

Some space groups allow valid combinations of WPs, but do not permit many (or any) positional degrees of freedom within the structure. It may also be the case that the allowed combinations result in atoms that are too close together. In these cases, PyXtal will attempt generation as usual: it will continue to search for a compatible structure until the maximum limit is reached, or until a successful generation occurs. In the event that structure generation repeatedly fails for a given combination of space group and stoichiometry, the user should make note and avoid the combination going forward.

2.1.2. Lattice generation

The first step in PyXtal's structure generation is the choice of unit cell. Depending on the symmetry group, a specific type of lattice must be generated. For all crystals, the conventional cell choice is used to avoid ambiguity. Lattice information can be predefined by the user in either vector form (a, b, c, α, β, γ) or in the form of a 3×3 matrix. If lattice information is not provided, PyXtal will attempt to estimate the volume based on the chemical composition, resulting in the generation of a random unit cell which satisfies the input constraints.

The most general case is the triclinic cell, from which other cell types can be obtained by applying certain constraints. To generate a triclinic cell, 3 real numbers are randomly chosen (using a Gaussian distribution centered at 0) as the off-diagonal values for a 3×3 shear matrix. By treating this shear matrix as a cell matrix, one obtains 3 lattice angles. For the lattice vector lengths, a random 3-vector between (0, 0, 0) and (1, 1, 1) is chosen (using a Gaussian distribution centered at (0.5, 0.5, 0.5)). The relative values of the x, y, and z coordinates are used for a, b, and c respectively and scaled based on the required volume. For other cell types, any free parameters are obtained using the same methods as for the triclinic case, and then constraints are applied. In the tetragonal case, for example, all angles will be fixed to 90 degrees. Thus, only a random vector is needed to generate the lattice constants.

For low-dimensional systems, not all three unit cell axes are periodic. Therefore, the algorithm must be altered slightly, as described below.

For the 2D case, we chose c to be the non-periodic axis by default. For layer groups 3–7 ($P112$, $P11m$, $P11a$, $P112/m$, $P112/a$), c is also the unique axis; for all other layer groups, a is the unique axis. The length of c (the crystal's "thickness") is an optional parameter which can be specified by the user. If no thickness is given, the algorithm will automatically compute a random value based on a Gaussian distribution centered at the cubic root of the estimated volume. In other words, c will have the same length as the other axes on average.

For the 1D case, c is the periodic axis by default. For rod groups 3–7 ($P221$, $Pm11$, $Pc11$, $P2/m11$, $P2/c11$), a is the unique axis; for all other rod groups, c is the unique axis. Instead of choosing a value for the thickness, we constrain the unit cell based on the cross-sectional area of the a–b plane. This area can be either specified by the user or generated randomly. As with the 2D and 3D cases, there is no preference for any axis to be longer or shorter than the others unless specified by the user.

For 0D clusters, we constrain the atoms to lie within either a sphere or an ellipsoid, depending on the point group. For spherically or polyhedrally symmetric point groups (C_1, C_i, D_2, D_{2h}, T, T_h, O, T_d, O_h, I, I_h), we define a sphere centered on the origin. For all other point groups (which have a unique rotational axis), we define an ellipsoid with its c-axis aligned with the rotational axis. The a- and b-axes are always of equal length to ensure rotational symmetry about the c-axis. The relative lengths for the ellipsoidal axes are chosen in the same way as for the 3D tetragonal case. In order for the 0D case to be compatible with the 1D, 2D, and 3D cases, we encode the spheres and ellipsoids as lattices (a cubic lattice for a sphere, or tetragonal lattice for an ellipsoid). Then, when generating atomic coordinates, we check whether the randomly chosen point lies within the sphere or ellipsoid. If not, we simply retry until it does.

2.1.3. Wyckoff position selection and merging

The central building block for crystals in PyXtal is the WP. Once a space group and lattice are chosen, WPs are inserted one at a time to add structure. In PyXtal, we closely follow the algorithm provided in Ref. [9] to place the atoms in different WPs. In general, PyXtal starts with the largest available WP, which is the general position of the symmetry group. If the number of atoms required is equal to or greater than the size of the general position, the algorithm proceeds. If fewer atoms are needed, the next largest WP (or set of WPs) is chosen, in order of descending multiplicity. This is done to ensure that larger positions are preferred over smaller ones; this reflects the greater prevalence of larger multiplicities seen in nature.

Once a WP is chosen, a random 3-vector between (0, 0, 0) and (1, 1, 1) is created. We call this the generating point for the WP. Using the closest projection of this vector onto the WP (the WP being a periodic set of points, lines, or planes), one obtains a set of coordinates in real space (the atomic positions for that WP). Then, the distances between these coordinates are checked. If the atom–atom distances are all greater than a pre-defined limit, the WP is kept and the algorithm continues. If any of the distances are too small, it is an indication that the WP would not occur with the chosen generating point. In this case, the coordinates are merged together into a smaller WP, if possible. This merging continues until the atoms are no longer too close together (see Fig. 2).

To merge into a smaller position, the original generating point is projected into each of the remaining WPs. The WP with the smallest translation between the original point and the transformed point is chosen, provided that (1) the new WP is a subset

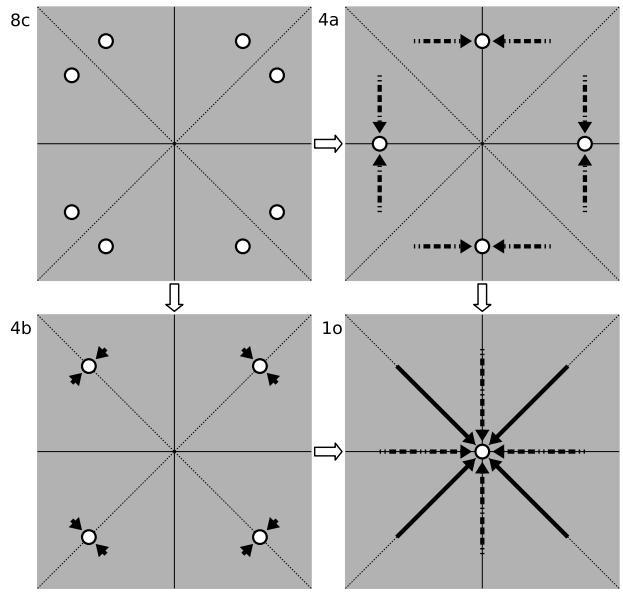

Fig. 2. Wyckoff Position Merging Example. Shown are possible merging of the general position 8c of the 2D point group 4 mm. Moving from 8c to 4b (along the solid arrows) requires a smaller translation than from 8c to 4a (along the dashed arrows). Thus, if the atoms in 8c were too close together, PyXtal would merge them into 4b instead of 4a. The atoms could be further merged into position 1o by following the arrows shown in the bottom right image.

of the original one, and (2) the new points are not too close to each other. If the atoms are still too close together after all possible mergings, the WP is discarded and another attempt is made.

Once a WP is successfully filled, the inter-atomic distances between the current WP and the already-added WPs are checked. If all distances are acceptable, the algorithm continues. More WPs are then added as needed until the desired number of atoms is reached. At this point, either a satisfactory structure has been generated, or the generation has failed. If the generation fails, then choosing either smaller distances tolerances or a larger volume factor might increase the chances of success. However, altering these quantities too drastically may result in less realistic crystals. Common sense and system-specific considerations should be applied when adjusting these parameters.

2.1.4. Distance checking

To produce structures with realistic bonds and bond lengths, the generated atoms should not be too close together. In PyXtal, this means that, by default, two atoms should be no closer than the covalent bond length between them. However, for a given application, the user may decide that shorter or longer cutoff distances are appropriate. For this reason, PyXtal has a custom *tolerance matrix* class which allows the user to define the distances allowed between any two atomic species. There are also options to use the metallic bond lengths, or to simply scale the allowed distances by some factor.

Because crystals have periodic symmetry, any point in a crystal actually corresponds to an infinite lattice of points. Likewise, any separation vector between two points actually corresponds to an infinite number of separation vectors. For the purposes of distance checking, only the shortest of these vectors is relevant. When a lattice is non-Euclidean, the problem of finding shortest distances with periodic boundary conditions is non-trivial, and the general solution can be computationally expensive [29]. So instead, PyXtal uses an approximate solution based on assumptions about the lattice geometry:

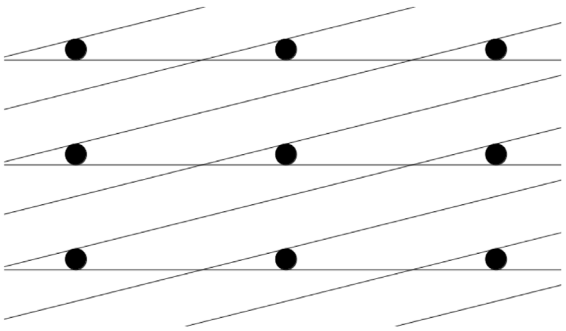

Fig. 3. Distorted Unit Cell. Due to the cell's high level of distortion, the closest neighbors for a single point lie more than two unit cells away. In this case, the closest point to the central point is located two cells to the left and one cell diagonal-up. To find this point using PyXtal's distance checking method, a $5 \times 5 \times 5$ unit cell will be created. For this reason, a limit is placed on the distortion of randomly generated lattices.

For any two given points, PyXtal first considers only the separation vector which lies within the "central" unit cell spanning between (0, 0, 0) and (1, 1, 1). For example, if the original two (fractional) points are (−8.1, 5.2, −4.8) and (2.7, −7.4, 9.3), one can directly obtain the separation vector (−10.8, 12.6, −14.1). This vector lies outside of the central unit cell, so we translate by the integer-valued vector (11.0, −12.0, 15.0) to obtain (0.2, 0.6, 0.9), which lies within the central unit cell. PyXtal also considers those vectors lying within a $3 \times 3 \times 3$ supercell centered on the first vector. In this example, these would include (1.2, 1.6, 1.9), (−0.8, −0.4, −0.1), (−0.8, 1.6, 0.9), etc. This gives a total of 27 separation vectors to consider. After converting to absolute coordinates (by dotting the fractional vectors with the cell matrix), one can calculate the Euclidean length of each of these vectors and thus find the shortest distance.

Note that this does not work for certain vectors within some highly distorted lattices (see Fig. 3). Often, the shortest Euclidean distance is accompanied by the shortest fractional distance, but whether this is the case or not depends on how distorted the lattice is. However, because randomly generated lattices in PyXtal are required to have no angles smaller than 30 degrees or larger than 150 degrees, this is not an issue.

For two given sets of atoms (for example, when cross-checking two WPs in the same crystal), one can calculate the shortest inter-atomic distances by applying the above procedure for each unique pair of atoms. This only works if it has already been established that both sets on their own satisfy the needed distance requirements.

Thanks to symmetry, one needs not calculate every atomic pair between two WPs. For two WPs, A and B, it is only necessary to calculate either (1) the separations between one atom in A and all atoms in B, or (2) one atom in B and all atoms in A. This is because the symmetry operations which duplicate a point in a WP also duplicate the separation vectors associated with that point. This is also true for a single WP; for example, in a Wyckoff position with 16 points, only 15 (the number of pairs involving one atom) distance calculations are needed, as opposed to 120 (the total number of pairs). This can significantly speed up the calculation for larger WPs.

For a single WP, it is necessary to calculate the distances for each unique atom–atom pair after symmetry reduction, but also for the lattice vectors for each atom by itself. Since the lattice is the same for all atoms in the crystal, this check only needs to be performed on a single atom of each species. For atomic crystals, this just means ensuring that the generated lattice vectors are sufficiently long.

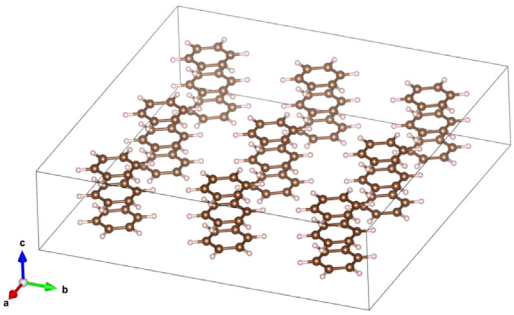

Fig. 4. Dependence of shortest distances on molecular orientation. Rotation of the molecules about the *a* or *b* (but not the *c*) axis would cause the benzene molecules to overlap. PyXtal checks for overlap whenever a molecular orientation is altered.

For molecules, the process is slightly more complicated. Depending on the molecule's orientation within the lattice, the inter-atomic distances can change. Additionally, one must calculate the distances not just between molecular centers, but between every unique atom–atom pair. This increases the number of needed calculations in rough proportion to the square of size of the molecules. As a result, this is typically the largest time cost for generation of molecular crystals. The issue of checking the lattice is also dependent on molecular orientation. Thus, the lattice must be checked for every molecular orientation in the crystal. To do this, the atoms in the original molecule are checked against the atoms in periodically translated copies of the molecule. Here, standard atom–atom distance checking is used. While several approximate methods for inter-molecular distance checking exist, their performance is highly dependent on the molecular shape and number of atoms. The simplest method is to model the molecule as a sphere, in which case only the center–center distances are needed. This works well for certain molecules, like buckminsterfullerene, which have a large number of atoms and are approximately spherical in shape. But a spherical model works poorly for irregularly shaped molecules like benzene (see Fig. 4), which may have short separations along the perpendicular axis, but must be further apart along the planar axes. We provide spherical distance checking as an option for the user, but direct atom–atom distance checking is used by default.

2.1.5. Molecular orientations

In crystallography, atoms are typically assumed to be spherically symmetric point particles with no well-defined orientation. Since the object occupying a crystallographic WP is usually an atom, it is further assumed that the object's symmetry group contains the WP's site symmetry as a subgroup. If this is the case, the only remaining condition for occupation of a WP is the location within the unit cell. However, if the object is instead a molecule, then the WP compatibility is also determined by orientation and shape.

To handle the general case, one must ensure that the object is (1) sufficiently symmetric, and is (2) oriented such that its symmetry operations are aligned with the Wyckoff site symmetry. The result is that objects with different point group symmetries are only compatible with certain WPs. For a given molecule and WP, one can find all valid orientations as follows:

1. Determine the molecule's point group and point group operations. This is currently handled by Pymatgen's built-in *PointGroupAnalyzer class* [30], which produces a list of symmetry operations for the molecule.

2. Associate an axis to every symmetry operation. For a rotation or improper rotation, we use the rotational axis. For a mirror plane, we use an axis perpendicular to the plane. Note that inversional symmetry does not add any constraints, since the inversion center is always located at the molecule's center of mass.

3. Choose up to two non-collinear axes from the site symmetry and calculate the angle between them. Find all conjugate operation pairs (with the same order and type) in the molecular point symmetry with the same angle between the axes, and store the rotation which maps the pairs of axes onto each other. For example, if the site symmetry were mmm, then we could choose two reflectional axes, say the x- and y-axes or the y- and z-axes. Then, we would look for two reflection operations in the molecular symmetry group. If the angle between these two operation axes is also 90 degrees, we would store the rotation that maps the two molecular axes onto the Wyckoff axes. We would also do this for every other pair of reflections with 90 degrees separating them.

4. For a given pair of axes, there are two rotations which can map one onto the other. There is one rotation which maps the first axis directly onto the second and another rotation which maps the first axis onto the opposite of the second axis. Depending on the molecular symmetry, the two resulting orientations may or may not be symmetrically equivalent. So, using the list of rotations calculated in step 3, remove redundant orientations which are equivalent to each other.

5. For each found orientation, check that the rotated molecule is symmetric under the Wyckoff site symmetry. To do this, simply check the site symmetry operations one at a time by applying each operation to the molecule and checking for equivalence with the untransformed molecule.

6. For the remaining valid orientations, store the rotation matrix and the number of degrees of freedom. If two axes were used to constrain the molecule, then there are no degrees of freedom. If one axis is used, then there is one rotational degree of freedom, and we store the axis about which the molecule may rotate. If no axes are used (because there are only point operations in the site symmetry), then there are three (stored internally as two) degrees of freedom, meaning the molecule can be rotated freely in 3 dimensions.

PyXtal performs these steps for every WP in the symmetry group and stores the nested list of valid orientations. When a molecule must be inserted into a WP, an allowed orientation is randomly chosen from this list. This forces the overall symmetry group to be preserved since symmetry-breaking WPs do not have any valid orientations to choose from. The above algorithm is particularly useful to generate molecular crystals with non-integer number of molecules in the asymmetric unit, which occur frequently for molecules with high point group symmetry.

One important consideration is whether a symmetry group will produce inverted copies of the constituent molecules. In many cases, a chiral molecule's mirror image will possess different chemical or biological properties [31]. For pharmaceutical applications in particular, one may not want to consider crystals containing mirror molecules. By default, PyXtal does not generate crystals with mirror copies of chiral molecules. The user can choose to allow inversion if desired.

2.2. Structure modulation

In many applications, it is often necessary to derive a new structure from the parent structure models. The main concept is to start from a simple, highly symmetrical crystal structure and to derive more complicated structures by applying a fairly small distortion or chemical substitution. Noticeable examples include non-reconstructive phase transitions and catalogue of structural prototype. In structure prediction methods such as simulated annealing and evolutionary algorithms, the generation

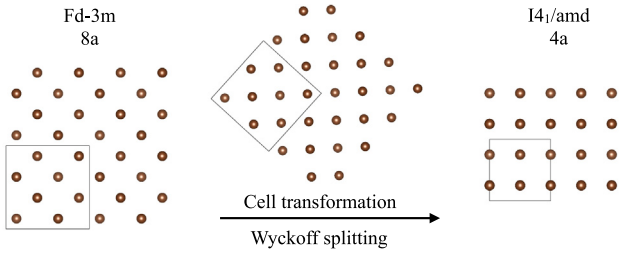

Fig. 5. A schematic example to illustrate group–subgroup transition from *Fd*-3*m* (8a) to $I4_1/amd$ (4a).

of new structures are usually obtained by applying a random perturbation on both lattice vectors and atomic coordinates. In this context, the introduction of symmetry relation can reduce the search space greatly. PyXtal provides two ways of structure manipulation based on the symmetry constraints.

2.2.1. Perturbation on asymmetric unit

The most straightforward way to preserve the crystal symmetry is to perturb only the atoms in the asymmetric unit. For atomic crystals, we only apply a random perturbation to each Wyckoff site if it has free coordinates. For molecular crystals, the whole molecule can be reoriented along the allowed rotational axes. If the molecule has flexible rotors, additional perturbation on these dihedral angles can be applied as well. Optionally, the cell parameters can be changed as well based on the symmetry constraints. Applying the symmetry-preserved mutation can be used to effectively search for molecular crystals in a constrained space group symmetry.

2.2.2. Group–subgroup transition

When one crystal is converted to another by a phase transition, the symmetries of the crystal structures are usually related. The so called group–subgroup has been well discussed mathematically. It is possible to list all possible subgroup types for every space-group type and to specify the subgroups in a general way by formulae. The international crystallography volume [32], as well as the online Bilbao Crystallographic Server [33], have provided the symmetry relations between a given space group G and its possible maximal translationengleiche (t) and klassengleiche (k) subgroups H. Ideally, to complete the transition from G to H, one needs to know the cell transformation matrix, as well as the Wyckoff splitting scheme. For a given crystal structure, PyXtal allows to either systematically extract all possible transformation (subject to a cutoff index of symmetry reduction), or randomly pick one possible transformation path between G and H. Fig. 5 shows an example to illustrate the transition from a structure occupying 8a in *Fd*-3*m* to another structure occupying 4a in $I4_1/amd$ symmetry.

3. Dependencies

All of the code is written in Python 3. Like many other Python packages, it relies on several external libraries. NumPy [34], SciPy [35] and Pandas [36] are required for the general purposes of scientific computing and data processing. In addition, two materials science libraries, Pymatgen [30] and Spglib [37], were used to facilitate the symmetry analysis. Optionally, the code provides an interface with Openbabel [38] if the users wants to import the molecules from additional file formats other than the plain xyz format. An ASE [39] interface is also enabled if the user wants to do further structure analysis such as structure manipulation or geometry optimization based on ASE.

4. Example usages

PyXtal can be either used as a binary executable or stand-alone library for use in Python scripts. Below we introduce the basic usages in brief.

4.1. Command line utilities

Currently, several utilities are available to access the different functionality of PyXtal. They include:

1. pyxtal_symmetry.py
2. pyxtal_main.py
3. pyxtal_test.py

First, the users are advised to run the pyxtal_test.py to quickly test if all modules are working correctly after the installation. The rest of the utilities are designed for different analysis purposes.

The pyxtal_symmetry.py utility allows one to easily access the symmetry information for a given symmetry group using either the group name or international number.

```
$ pyxtal_symmetry.py -s 64
-- Spacegroup --# 64 (Cmce)--
16gsite symm: 1
8fsite symm: m..
8esite symm: .2.
8dsite symm: 2..
8csite symm: -1
4bsite symm: 2/m..
4asite symm: 2/m..
```

Listing 1: Example usage of the pyxtal_symmetry.py utility.

The pyxtal_main.py can be used to directly generate one trial structure based on the given symmetry group and chemical composition. Below, we give the example scripts to generate different types of symmetric objects, including

1. a random C60 cluster with I_h point group symmetry;
2. a trial diamond structure with *Fd-3m* space group symmetry;
3. a crystal of two C60 molecules per primitive unit cell with $Cmc2_1$ symmetry

```
$ pyxtal_main.py -e C -n 60 -d 0 -s Ih
$ pyxtal_main.py -e C -n 2 -s 227
$ pyxtal_main.py -m -e C60 -n 4 -s 36
```

Listing 2: Example usages of the pyxtal_main utility.

The generated structures will be saved to text files in cif format for crystals and xyz format for clusters.

4.2. Structure modulation and analysis

In addition to structure generation and manipulation, PyXtal also provides several other utilities, such as XRD analysis. Below, we give the example scripts to (1) generate a cubic crystal, (2) perturb the structure to lower the crystal symmetry by following the group–subgroup relation, and (3) compare two PXRD between two structures. Finally, Fig. 6 displays the simulated XRDs between two structures, where the similarity between two XRDs are also given according to the correlation function as suggested previously [40,41]. Note the split of XRD peaks.

```
from pyxtal import pyxtal
from pyxtal.XRD import Similarity

# generate a random crystal
C1 = pyxtal()
C1.from_random(3, 227, ['C'], [8])
```

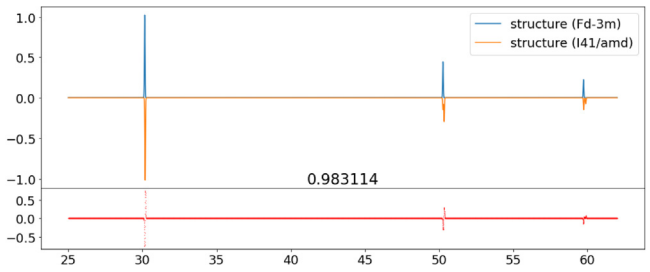

Fig. 6. The comparison of simulated X-ray diffraction patterns between the original structure and the perturbed structure from the group–subgroup relation.

```
# perturbation without changing the symmetry
# C2 = C1.copy()
# C2.apply_perturbation()

# lower the symmetry from cubic to tetragonal
C2 = C1.subgroup_once(H=141)

# Compute the XRD
xrd1 = C1.get_XRD()
xrd2 = C2.get_XRD()

# Compare two structures by XRD
p1 = xrd1.get_profile()
p2 = xrd2.get_profile()
s = Similarity(p1, p2, x_range=[15, 90])
s.show(filename='xrd-comparison.png')
```

Listing 3: Example usages of PyXtal's structural manipulation and analysis utilities

4.3. Structure prediction

PyXtal allows the user to generate random crystal structures with given symmetry constraints. There are several parameters which can be specified, but only a few are necessary. Below is an example script to generate 100 random clusters for 36 carbon atoms.

```
from pyxtal import pyxtal
from random import choice

pgs = range(1, 33)
clusters = []
for i in range(100):
    while True:
        pg = choice(pgs)
        cluster = pyxtal()
        cluster.from_random(0, pg, ['C'], [36],
    force_pass=True)
        if cluster.valid:
            clusters.append(cluster)
            break
```

Listing 4: A Python script to generate 100 random C36 clusters

With the generated structures, one can perform further analysis such as geometry optimization and powder X-ray diffraction pattern simulation. PyXtal also provides the preliminary modules for such tasks. Alternatively, the trial structures can be easily adapted to the structural objects for other libraries, such as ASE [39] or Pymatgen [30], or be dumped to text files in cif, xyz, or POSCAR format. More examples can be found in the online documentation https://pyxtal.readthedocs.io.

5. Applications

Our primary purpose for developing PyXtal is to provide more likely trial structures to solve the structural determination problem. It can be useful for at least two cases. First, one can generate the trial structures based on the partial information determined from experiment (e.g., unit cell, symmetry, composition). Secondly, it can be used to determine the ground state structure in a first-principle manner based on global optimization. It has been shown [6,10,12] that by beginning with already-symmetric structures, fewer attempts are needed to find the global energy minimum. To demonstrate the general utility of pre-symmetrization, we performed a number of benchmarks for different systems. Below we give two examples for the global structural search on the low-energy Lennard-Jones (LJ) clusters and carbon/silicon allotropes.

5.1. Clusters with empirical Lennard-Jones potential

Finding the ground state of LJ clusters of given size is an established benchmark for global optimization methods [5]. Here, it shows that local optimization, combined with randomly generated symmetric clusters, is sufficient to solve the problem with small sizes of LJ clusters. For the purposes of this benchmark, we focus on three cluster sizes, namely 38, 55, and 75. For each cluster size, 20,000 structures were generated: 10,000 with no pre-defined symmetry and 10,000 with symmetry chosen randomly from among PyXtal's 56 built-in point groups. A potential of $4(\frac{1}{r^{12}} - \frac{1}{r^6})$ was assigned to each atom–atom pair. Each structure was locally optimized using the conjugate gradient (CG) method in SciPy's *optimize.minimize* function [35]. As shown in Fig. 7, the ground state was found much more frequently when the initial structures possessed some point group symmetry. With pre-symmetrization, the ground state was found 278 times for size 38 clusters, 73 times for size 55, and 1 time for size 75. Without pre-symmetrization, the ground state was not found at all. Though the numbers of hits on the ground states may change in another run, the statistical rule still holds. Second, while the ground state is found more frequently with pre-symmetrization, the average energy is higher. This is because pre-symmetrization spans the possible structure space more effectively, while purely random structures are more clustered around a specific energy range.

5.2. Carbon and silicon crystals with ab-initio calculations

We also combined PyXtal with ab-initio codes to search for the elemental allotropes of carbon and silicon at 0 K and ambient pressure. 1000 random structures each were generated for 2, 4, 6, 8, and 16 atoms in the primitive unit cell. A random space group between 2 and 230 was chosen for each structure. This gave a total of 5000 structures for each element. Each structure was optimized using the PBE–GGA functional [42] as implemented in the VASP code [43,44], following a multiple-step strategy from low, normal, to accurate precision. The final geometries were then calculated with an energy cutoff of 600 eV and 0.15 K-spacing. For carbon, the expected structures of diamond and graphite were found frequently in each run, as well as londsdaelite, $sp3$ carbon with various ring topologies, and various multi-layer graphite-like structures. Similarly, our simulation on silicon yielded the ground state of cubic diamond structures for each of the runs with different numbers of atoms per primitive unit cell, demonstrating that adding symmetry constraints is beneficial to quickly identify the low-energy structures with high symmetry. Moreover, it is again interesting to analyze the energy distribution of the randomly generated structures as shown in Fig. 8. For both carbon and silicon, the energy landscape appears to be narrower for size-2 primitive cells. It appears that beyond about 4, the number of atoms in the primitive cell has little influence on the energy distribution. This again suggests that pre-symmetrization is an effective means to prevent the clustering of glassy structures found

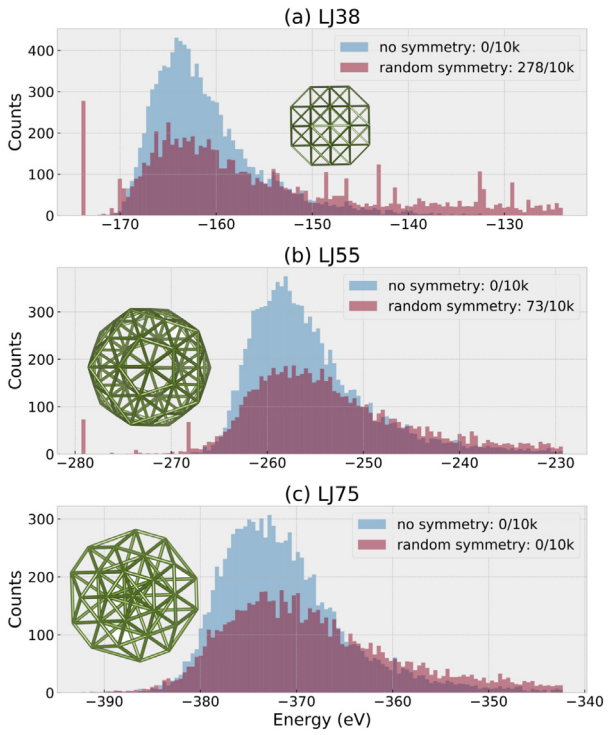

Fig. 7. Energy distribution for Lennard Jones clusters with the sizes of (a) 38, (b) 55 and (c) 75. The insets are the corresponding ground state geometries.

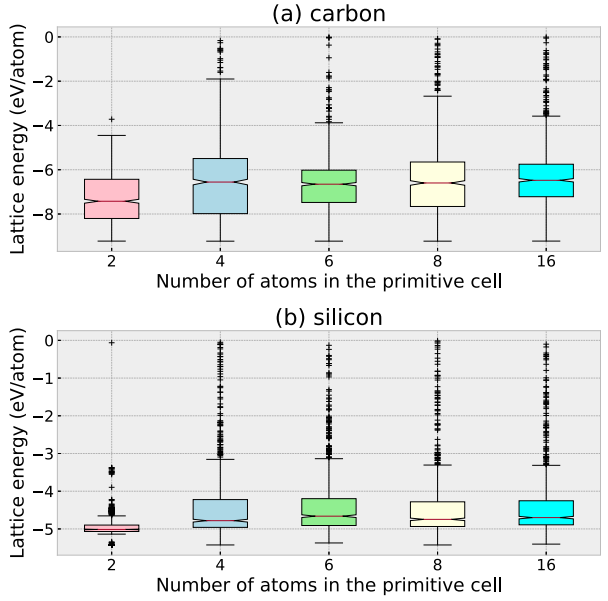

Fig. 8. The box and whisker plots for the energy distribution of the randomly generated (a) carbon and (b) silicon crystals with 2, 4, 6, 8, 16 atoms per primitive unit cell.

in pure random generations for large systems [9]. Therefore, pre-symmetrization provides a better choice for global energy optimization. In addition, pre-symmetrization can provide a more diverse dataset for training machine learning force fields [45,46].

6. Conclusion

In this manuscript, we present a software package PyXtal. The core features of PyXtal have been highlighted, with further documentation available online.[1] In PyXtal, the symmetry constraints are further refined in three ways. The first is a merging algorithm [14] which controls the distribution of WPs through statistical means. The second is a new algorithm for placing molecules into special WPs. Last is the structure modulation by the observation of the symmetry relation. This allows for more realistic and complex structures to be generated by keeping the symmetry as high as possible. PyXtal is not a complete structure prediction package; it only generates the trial structures with a given symmetry group. Other tools exist that perform structure generation and other steps in the CSP process [3,7,9,10]. The main goals in developing PyXtal are as follows: (1) to develop a free, open-source Python package for the materials science community, (2) to handle the generation of symmetric structures described by different symmetry groups from 0D to 3D, (3) to handle molecular WPs in a generalized manner, (4) to provide a tool to analyze the symmetry relation. We also demonstrated that using the pre-symmetrized structures as the starting seeding structures can effectively improve the success rate of finding the low energy configuration. As such, PyXtal can be interfaced with other structure prediction codes that require the generation of trial structures. Access to the source code and development information are available on the GitHub page at https://github.com/qzhu2017/PyXtal. The code is currently under version 0.1.7 at the time of writing. It is expected to update frequently. Further development and application of the mathematical background should enable more complex structure types to be studied in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We acknowledge the NSF, USA (I-DIRSE-IL: 1940272) and NASA, USA (80NSSC19M0152) for their financial supports. The computing resources are provided by XSEDE (TG-DMR180040).

References

[1] A.R. Oganov, Modern Methods of Crystal Structure Prediction, John Wiley & Sons, 2011, http://dx.doi.org/10.1002/9783527632831.
[2] A.R. Oganov, C.J. Pickard, Q. Zhu, R.J. Needs, Nat. Rev. Mater. 4 (2019) 331–348, http://dx.doi.org/10.1038/s41578-019-0101-8.
[3] C.J. Pickard, R. Needs, J. Phys. Condens. Matter 23 (5) (2011) 053201, http://dx.doi.org/10.1088/0953-8984/23/5/053201.
[4] R. Martoňák, A. Laio, M. Parrinello, Phys. Rev. Lett. 90 (7) (2003) 075503, http://dx.doi.org/10.1103/PhysRevLett.90.075503.
[5] D.J. Wales, J.P.K. Doye, J. Phys. Chem. A 101 (28) (1997) 5111–5116, http://dx.doi.org/10.1021/jp970984n.
[6] A.R. Oganov, C.W. Glass, J. Chem. Phys. 124 (24) (2006) 244704, http://dx.doi.org/10.1063/1.2210932.
[7] D.C. Lonie, E. Zurek, Comput. Phys. Comm. 182 (2) (2011) 372–387, http://dx.doi.org/10.1016/j.cpc.2010.07.048.
[8] Y. Wang, J. Lv, L. Zhu, Y. Ma, Phys. Rev. B 82 (9) (2010) 094116, http://dx.doi.org/10.1103/PhysRevB.82.094116.
[9] A.O. Lyakhov, A.R. Oganov, H.T. Stokes, Q. Zhu, Comput. Phys. Comm. 184 (4) (2013) 1172–1182, http://dx.doi.org/10.1016/j.cpc.2012.12.009.
[10] Y. Wang, J. Lv, L. Zhu, Y. Ma, Comput. Phys. Comm. 183 (10) (2012) 2063–2070, http://dx.doi.org/10.1016/j.cpc.2012.05.008.
[11] H. Wondratschek, M.I. Aroyo, Int. Tables Crystallogr. A (2016) 12–21.
[12] P. Avery, E. Zurek, Comput. Phys. Comm. 213 (2017) 208–216, http://dx.doi.org/10.1016/j.cpc.2016.12.005.
[13] R. Domingos, K.M. Shaik, B. Militzer, Phys. Rev. B 98 (2018) 174107, http://dx.doi.org/10.1103/PhysRevB.98.174107.

[1] https://pyxtal.readthedocs.io.

[14] Q. Zhu, A. Oganov, C. Glass, H.T. Stokes, Acta Crystallogr. Sect. B 68 (2012) 215–26, http://dx.doi.org/10.1107/S0108768112017466.
[15] A.M. Reilly, R.I. Cooper, C.S. Adjiman, S. Bhattacharya, A.D. Boese, J.G. Brandenburg, P.J. Bygrave, R. Bylsma, J.E. Campbell, R. Car, D.H. Case, R. Chadha, J.C. Cole, K. Cosburn, H.M. Cuppen, F. Curtis, G.M. Day, R.A. DiStasio Jr, A. Dzyabchenko, B.P. van Eijck, D.M. Elking, J.A. van den Ende, J.C. Facelli, M.B. Ferraro, L. Fusti-Molnar, C.-A. Gatsiou, T.S. Gee, R. de Gelder, L.M. Ghiringhelli, H. Goto, S. Grimme, R. Guo, D.W.M. Hofmann, J. Hoja, R.K. Hylton, L. Iuzzolino, W. Jankiewicz, D.T. de Jong, J. Kendrick, N.J.J. de Klerk, H.-Y. Ko, L.N. Kuleshova, X. Li, S. Lohani, F.J.J. Leusen, A.M. Lund, J. Lv, Y. Ma, N. Marom, A.E. Masunov, P. McCabe, D.P. McMahon, H. Meekes, M.P. Metz, A.J. Misquitta, S. Mohamed, B. Monserrat, R.J. Needs, M.A. Neumann, J. Nyman, S. Obata, H. Oberhofer, A.R. Oganov, A.M. Orendt, G.I. Pagola, C.C. Pantelides, C.J. Pickard, R. Podeszwa, L.S. Price, S.L. Price, A. Pulido, M.G. Read, K. Reuter, E. Schneider, C. Schober, G.P. Shields, P. Singh, I.J. Sugden, K. Szalewicz, C.R. Taylor, A. Tkatchenko, M.E. Tuckerman, F. Vacarro, M. Vasileiadis, A. Vazquez-Mayagoitia, L. Vogt, Y. Wang, R.E. Watson, G.A. de Wijs, J. Yang, Q. Zhu, C.R. Groom, Acta Crystallogr. Sect. B 72 (4) (2016) 439–459, http://dx.doi.org/10.1107/S2052520616007447.
[16] W.H. Baur, D. Kassner, Acta Crystallogr. Sect. B 48 (4) (1992) 356–369, http://dx.doi.org/10.1107/S0108768191014726.
[17] B.P. van Eijck, J. Kroon, J. Comput. Chem. 20 (8) (1999) 799–812, http://dx.doi.org/10.1002/(SICI)1096-987X(199906)20:8<799::AID-JCC6>3.0.CO;2-Z.
[18] J.R. Holden, Z. Du, H.L. Ammon, J. Comput. Chem. 14 (4) (1993) 422–437, http://dx.doi.org/10.1002/jcc.540140406.
[19] D.H. Case, J.E. Campbell, P.J. Bygrave, G.M. Day, J. Chem. Theory Comput. 12 (2) (2016) 910–924, http://dx.doi.org/10.1021/acs.jctc.5b01112.
[20] M.A. Neumann, F.J. Leusen, J. Kendrick, Angew. Chem. Int. Ed. 120 (13) (2008) 2461–2464, http://dx.doi.org/10.1002/anie.200704247.
[21] F. Curtis, X. Li, T. Rose, Á. Vázquez-Mayagoitia, S. Bhattacharya, L.M. Ghiringhelli, N. Marom, J. Chem. Theory Comput. 14 (4) (2018) 2246–2264, http://dx.doi.org/10.1021/acs.jctc.7b01152.
[22] R. Tom, T. Rose, I. Bier, H. O'Brien, A. Vazquez-Mayagoitia, N. Marom, Comput. Phys. Comm. 250 (2020) 107170, http://dx.doi.org/10.1016/j.cpc.2020.107170.
[23] A.G. Shtukenberg, C.T. Hu, Q. Zhu, M.U. Schmidt, W. Xu, M. Tan, B. Kahr, Cryst. Growth Des. 17 (6) (2017) 3562–3566, http://dx.doi.org/10.1021/acs.cgd.7b00673.
[24] Q. Zhu, A.G. Shtukenberg, D.J. Carter, T.-Q. Yu, J. Yang, M. Chen, P. Raiteri, A.R. Oganov, B. Pokroy, I. Polishchuk, P.J. Bygrave, G.M. Day, A.L. Rohl, M.E. Tuckerman, B. Kahr, J. Am. Chem. Soc. 138 (14) (2016) 4881–4889, http://dx.doi.org/10.1021/jacs.6b01120.
[25] A.G. Shtukenberg, Q. Zhu, D.J. Carter, L. Vogt, J. Hoja, E. Schneider, H. Song, B. Pokroy, I. Polishchuk, A. Tkatchenko, et al., Chem. Sci. 8 (7) (2017) 4926–4940, http://dx.doi.org/10.1039/C7SC00168A.
[26] W. Xu, Q. Zhu, C.T. Hu, Angew. Chem. Int. Ed. 129 (8) (2017) 2062–2066, http://dx.doi.org/10.1002/ange.201610977.
[27] J. Yang, C.T. Hu, X. Zhu, Q. Zhu, M.D. Ward, B. Kahr, Angew. Chem. Int. Ed. 56 (34) (2017) 10165–10169, http://dx.doi.org/10.1002/anie.201703028.
[28] M. Tan, A. Shtukenberg, S. Zhu, W. Xu, E. Dooryhee, S.M. Nichols, M.D. Ward, B. Kahr, Q. Zhu, Roy revisited, again: The eighth solved structure, Faraday Discuss, http://dx.doi.org/10.1039/C8FD00039E.
[29] L. Babai, Combinatorica 6 (1) (1986) 1–13, http://dx.doi.org/10.1007/BF02579403.
[30] S.P. Ong, W.D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V.L. Chevrier, K.A. Persson, G. Ceder, Comput. Mater. Sci. 68 (2013) 314–319, http://dx.doi.org/10.1016/j.commatsci.2012.10.028.
[31] I.-H. Suh, K.H. Park, W.P. Jensen, D.E. Lewis, J. Chem. Educ. 74 (7) (1997) 800, http://dx.doi.org/10.1021/ed074p800.
[32] H. Wondratschek, U. Müller, Int. Union. Crystallogr. (2006) http://dx.doi.org/10.1107/97809553602060000110.
[33] M.I. Aroyo, J.M. Perez-Mato, C. Capillas, E. Kroumova, S. Ivantchev, G. Madariaga, A. Kirov, H. Wondratschek, Z. Kristallogr. Cryst. Mater. 221 (1) (2006) 15–27, http://dx.doi.org/10.1524/zkri.2006.221.1.15.
[34] T.E. Oliphant, A Guide To NumPy, 1, Trelgol Publishing, USA, 2006.
[35] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberl and, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E.W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python, ArXiv E-Prints, S... Contributors, 2019, arXiv:1907.10121.
[36] W. McKinney, Data structures for statistical computing in python, in: S. van der Walt, J. Millman (Eds.), Proceedings of the 9th Python in Science Conference, 2010, pp. 51–56.
[37] A. Togo, I. Tanaka, Spglib: A Software Library for Crystal Symmetry Search, 2018, arXiv:1808.01590.
[38] R. Guha, M.T. Howard, G.R. Hutchison, P. Murray-Rust, H. Rzepa, C. Steinbeck, J.K. Wegner, E. Willighagen, J. Chem. Inf. Model. 46 (2006) 991, http://dx.doi.org/10.1021/ci050400b.
[39] A.H. Larsen, J.J. Mortensen, J. Blomqvist, I.E. Castelli, R. Christensen, M. Dułak, J. Friis, M.N. Groves, B. Hammer, C. Hargus, E.D. Hermes, P.C. Jennings, P.B. Jensen, J. Kermode, J.R. Kitchin, E.L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J.B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K.S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, K.W. Jacobsen, J. Phys. Conden. Matter 29 (27) (2017) 273002, http://dx.doi.org/10.1088/1361-648x/aa680e.
[40] R. de Gelder, R. Wehrens, J.A. Hageman, J. Comput. Chem. 22 (3) (2001) 273–289, http://dx.doi.org/10.1002/1096-987X(200102)22:3<273::AID-JCC1001>3.0.CO;2-0.
[41] S. Habermehl, P. Mörschel, P. Eisenbrandt, S.M. Hammer, M.U. Schmidt, Crystal Engineering and Materials 70 (2) (2014) 347–359, http://dx.doi.org/10.1107/S2052520613033994.
[42] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865–3868, http://dx.doi.org/10.1103/PhysRevLett.77.3865.
[43] G. Kresse, J. Furthmüller, Comput. Mater. Sci. 6 (1996) 15–50, http://dx.doi.org/10.1016/0927-0256(96)00008-0.
[44] G. Kresse, J. Furthmüller, Phys. Rev. B 54 (1996) 11169–11186, http://dx.doi.org/10.1103/PhysRevB.54.11169.
[45] V.L. Deringer, C.J. Pickard, G. Csányi, Phys. Rev. Lett. 120 (2018) 156001, http://dx.doi.org/10.1103/PhysRevLett.120.156001.
[46] E.V. Podryabinkin, E.V. Tikhonov, A.V. Shapeev, A.R. Oganov, Phys. Rev. B 99 (2019) 064114, http://dx.doi.org/10.1103/PhysRevB.99.064114.